Toward Scalable Blockchains with Transaction Aggregation

Niya, Sina Rafati ; Maddaloni, Fabio ; Bocek, Thomas ; Stiller, Burkhard

Abstract: Blockchains (BCs) are back-linked chain of records termed as blocks. To establish decentralized trusted systems, BCs employ consensus mechanisms. During the past ten years, there have been various proposals of BC design and implementations. However, most of the developed sate of the art BCs suffer from scalability issues. In order to enhance the scalability of the BCs, this paper proposes a transaction aggregation mechanism on a Proof-of-Stake (PoS)-based BC. Having developed the transaction aggregation and double linked blocks, efficient prevention and control of the BC's size growth is observed in the evaluated scenarios.

DOI: https://doi.org/10.1145/3341105.3373899

Toward Scalable Blockchains with Transaction Aggregation

***Abstract*—Blockchains (BCs) are back-linked chain of records termed as blocks. To establish decentralized trusted systems, BCs employ consensus mechanisms. During the past ten years, there have been various proposals of BC design and implementations. However, most of the developed sate of the art BCs suffer from scalability issues. In order to enhance the scalability of the BCs, this paper proposes a transaction aggregation mechanism on a Proof-of-Stake (PoS)-based BC. Having developed the transaction aggregation and double linked blocks, efficient prevention and control of the BC's size growth is observed in the evaluated scenarios.**

***Index Terms*—Scalable Blockchains, Proof-of-Stake (PoS), Transaction Aggregation**

I. Introduction

Blockchains (BCs) are distributed and decentralized data storage in terms of records within back-linked lists of blocks. Blockchains are obliged to offer sophisticated consensus methods to support trust for different applications. The most important characteristics of consensus algorithms include scalability, transaction rate, transmission delay, power consumption, security, and privacy, which determine very practical dimensions of BCs and their applicability for real applications [1].

The most used and prominent consensus mechanism is the Proof-of-Work (PoW) of the Bitcoin BC [2]. PoW is the proof of the attempt each miner puts to mine a new block. The attempt to find a new nonce, with which the hash function produces the output in the range of a requested target is the outcome. This effort is needed as the acceptable output may be reached after many iterations of the hashing function being applied to solve a cryptographic hash puzzle [3].

The second most used consensus mechanism is the Proof-of-Stake (PoS) [4]. PoS determines a category of consensus algorithms, which provide significant improvements in terms of electricity consumption and scalability over PoW. In a PoS consensus any node in the system can become a validator by depositing tokens of an associated cryptocurrency, which is different compared to PoW.

State of the art approaches of enhancing scalability of the PoS-based BCs such as sharding, Raiden, and Plasma [5] can be distinguished based on the employed approaches. One type of scalability enhancement methods is to aggregate transactions which can be combined by most of the other approaches to reduce the size of the BC. By reducing the BC size, the inter miner synchronizations specially when a new miner arrives to the network happens more efficiently. A comprehensive study on BCs and their approaches for enhancing scalability issues is presented in [6] and [7].

The idea behind transaction aggregation is to aggregate valid transactions such that multiple transactions from one sender or to one receiver are visible as one transaction in the BC. This should enhance the amount of *TPS* because more transactions can be validated in one block. Thus, the block size becomes less of a limiting factor when many transactions with the same sender or receiver are issued to the network. Furthermore, transaction aggregation reduces the BC's overall size because fewer transactions are visible in the BC. Especially when aggregating transactions, which are validated in an already closed block, the overall BC size can shrink since at a certain point these blocks can be emptied completely. This results in reduced block size.For the aggregation, a new type of transactions, called *AggTx*, is introduced. Furthermore, all funds transactions are updated accordingly.

This work here presents design and evaluation of a novel transaction aggregation mechanism facilitated by double linked blocks. The approaches introduced here follow generic concepts of PoS-based BCs, thus can be applied to a great variety of PoS-based BCs.

This paper is organized as follows, Sec. I-A explains the assumptions in this paper. Section II introduces the designed transaction aggregation algorithm. Section III discusses the evaluation results while the obstacles of transaction aggregation are presented in Sec. IV. At the end of this paper, conclusions are presented in the Sec. V.

A. Assumptions and Considerations

This work is based on a PoS-based BC which is nor named neither referenced to the code on Github to adhere the double blind review processes. Thus, after this, the employed BC is named as "Test-BC". The Test-BC chooses validators proportionally to the number of coins that each validator owns.

While the related work is well studied for this work, due to the lack of space, it is considered that interested readers will access the overviews of BCs and the scalability mechanisms employed in each can form [6] and [7] or other relevant references.

II. Scalability Enhancements of PoS Blockchains

A simple example of aggregating transactions can be thought as follows: User *A* sends 2Coins to *B* and 5 Coins to *C*. Without transaction aggregation both transactions are listed in a block as *(A →B : 2) & (A →C : 5)*. In contrast, with transaction aggregation, only one transaction shall be written into the block *(A →[B,C] : 7)*. Thus the overall BC size could be smaller and more transactions can be handled since they are

aggregated. The aggregation process introduced in this work is designed to be fully hidden from the users.

A. Transaction Aggregation (AggTx) Design

This work here designs a new transaction i.e., *AggTx*, specifically to aggregate and sum up the matching *funds* transactions and older aggregation transactions. Instead of these transactions, an *AggTx* will be listed inside a block. The *AggTx* is added to the blocks similar to other transactions. They are listed in the $AggTxData$ slice. The fields inside a *AggTx* is considered as following.

Amount: The amount is the summed up amount of all transactions aggregated inside this *AggTx*.

Fee: The fee of this transaction. It is set to 0 because, at the time of writing, the users should not be charged for this type of transaction.

From: It is a slice where the addresses of all senders sending a transaction aggregated in this block are stored.

To: This is the counterpart of the *From* field and filled with the addresses of the transactions' receivers.

AggregatedTxSlice: This slice is of type *[][32] Byte* and does store all hashes of the transactions aggregated inside this *AggTx*.

Aggregated: a *boolean* variable that indicates if the transaction is aggregated.

Block: In this filed the hash of the block – in which this transaction is aggregated the first time– stored.

MerkleRoot: Root of the Merkle tree to ensure integrity and the correct order for the transactions aggregated in this *AggTx*.

B. Theoretical Transaction Aggregation Processes

Funds transactions and aggregation transactions can be aggregated in two different ways. All other transactions types are not taken into account. Furthermore, it is not possible to combine an already aggregated transaction aggregated by the sender and one by the receiver. This results in either the *From* or *To* slice to have a length of one.

If transactions aggregated is done based on the sender address, all the transactions which are sent by that specific wallet are aggregated into one *AggTx*. Hence, the *From* slice has a length of one, as there is only one sender included. On the other hand, transactions can also be aggregated by the receiver and all transactions sent to one specific wallet are aggregated into one *AggTx*. Here the *To* slice is only of the length one since all transactions are sent to one specific receiver.

When a miner combs through all open transactions he tries to aggregate as many open funds transactions as possible according to these two rules. If two or more transactions can be aggregated, their transaction hashes are written to the *AggTx's AggregatedTxSlice* and the transactions' boolean *Aggregated* will be set to *true*.

In the next step, the miner checks already closed blocks, whether there are transactions (either *FundsTx* or *AggTx*) which match the chosen pattern (either aggregated by sender or receiver) and are not aggregated until now. If such historic transactions exist, they are also added to the *AggTx*. But they do not have an influence on the BC state during the post validation of a block anymore.

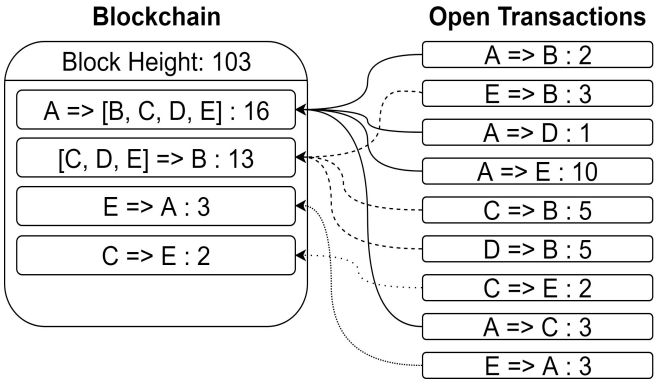


Fig. 1: Transaction Aggregation Concept

In Fig. 1 the aggregation process is visualized schematically. The letters are wallets and the numbers are the amount of coins sent. All open transactions are listed on the right side. In this example, the historic aggregation is omitted due to simplicity and only one of various possibilities is shown.

Algorithm 1 is designed to group the transactions in an optimal way, such that the least transactions are listed in the block, but the most transactions are validated. As shown in Fig. 1 without transaction aggregation, all these open transactions try to be in current block 103. When the block size is assumed to be limited to five transactions, with aggregation, there is still place for one more transaction whereas without transaction aggregation not even all nine transactions can be validated in the current block. Because Test-BC only writes the hashes of transactions inside its blocks, this is possible. Consequently, with aggregation only the hashes of the two aggregated transactions and the two normal funds transactions, which cannot be aggregated in this block, are stored in the block's body.

Furthermore, it is also visible that it actually does not matter, how many transactions are aggregated in one *AggTx*. If *A* would have sent more transactions, still only one transaction is written into the block, but this transaction would aggregate more transactions. This is especially nice for a BC which is used in a case, where often multiple transactions are sent from one or to one peer. With small modifications, an imaginable example use case would be a BC which stores values sent from Internet-Of-Things devices always to the same receiver.

In this design, miners do not earn a specific fee for aggregating. But they still receive all the fees which belong to the found transactions i.e., *FundsTx*. This results in miners that want to validate as many *FundsTx* as possible and thus earning as much as feasible. The more transactions they can aggregate the more transactions are in a block and they will get a higher reward. Since the block's size does not grow with every transaction, they can add more transaction into one block.

Since only valid *FundsTx* and already validated *AggTx* can be aggregated, the aggregation takes place after a *FundsTx* is characterized as a valid transaction. Instead of adding this valid transaction directly into the block's body it gets added into a temporary slice. Transactions in this slice will be sorted by the sender's address and then by the transaction counter.

All different senders and receivers are stored into two maps. The number of occurrences in all open transactions, which can possibly be aggregated, are used as values. These maps are called *diffS* and *diffR* in algorithm 1. This approach finds the best combination of how transactions will be aggregated (either by the sender or by receiver address). The *getMaxSenderReceiver(diffS, diffR)* function returns the sender and receiver with the most occurrences.

Algorithm 1 groups transactions in a way that open transactions either have the same sender or the same receiver. The output is then a new slice of transactions which matches the rules defined for transaction aggregation. The selected transactions are stored into the $txToAggregate$ slice and aggregated with the function on line 19. Then the algorithm removes this group of transactions from $possibleTxToAggregate$ and recalculates the *diffS* and *diffR*. A miner repeats these steps until *possibleTxToAggregate* is empty. This ensures, that the highest maximal number of transactions is validated in a block.

Algorithm 1: Split valid transactions for aggregation: splitSortedAggregatableTransactions *(block b)*

```
Input  : block b
Output: Slice of transactions which can be aggregated and block b, into
        which they belong

[1]  get possibleTxToAggregate from TempList
[2]  get diffS and diffR
[3]  sort possibleTxToAggregate by senderAddress and then TxCnt
[4]  for moreTxToAggregate do
[5]    maxSender, maxReceiver := getMaxSenderReceiver(diffS, diffR)
[6]    if maxSender >= maxReceiver then
[7]      forall tx in possibleTxToAggregate do
[8]        if tx.From == maxSender then append tx to txToAggregate
[9]        else keep tx in possibleTxToAggregate
[10]     end
[11]   else
[12]     forall tx in possibleTxToAggregate do
[13]       if tx.From == maxSender then append tx to txToAggregate
[14]       else keep tx in possibleTxToAggregate
[15]     end
[16]   end
[17]   remove txToAggregate from possibleTxToAggregate
[18]   recount diffS & diffR
[19]   AggregateTransactions(txToAggregate, b)
[20]   set txToAggregate to nil
[21]   if len(possibleTxToAggregate) == 0 then moreTxToAggregate = false
[22]   else moreTxToAggregate = true
[23] end
```

C. Double Linked Blockchain

The concept of a double linked BC is a result of the idea to remove all transactions from a block once all of them are aggregated in a later validated block. This is kind of a contradiction against the theory of a BC where all validated blocks are immutable. They are unchangeable because it would take too much effort to recalculate the complete chain since this adapted block, and additionally persuade over 50% of all miners to accept the newly created blocks.

In Test-BC the block hash is calculated from various block related input fields. One of these variables is the *Merkle root*, which ensures transaction verification. It ascertains that transactions neither can be added to or removed from a block nor the ordering can be changed once a block hash is created. Thus, removing of transactions is only possible when a new additional block hash is calculated for every block because the old hash is becoming invalid, as soon as some transactions are removed. This new hash, called *HashWithoutTransactions*, is used always when the normal hash is becoming invalid. The normal hash becomes invalid because the *Merkle root* changes.

The goal and also the specification of the double linking is, that at least one link to the previous block is valid. Additionally to the common variables included in a block, the following fields are added in respect of the double linking of the BC:

Aggregated: It indicates if a block is aggregated and therefore does not contain any transactions anymore. It is of type *boolean*.

HashWithoutTransactions: This hash is used once all transactions from a specific block are aggregated. It can be calculated when not taking the transactions into account and as a consequence assuming an empty block. Thus, it is only possible to empty a block, once all transactions are aggregated and removed. It is of type *[32]byte*.

PrevHashWithoutTransactions: This field links the current block to the previous one once all transactions in the previous block are aggregated.

ConflictingBlockHashWithoutTx1: HashWithoutTransactions of the first conflicting block.

ConflictingBlockHashWithoutTx2: HashWithoutTransactions of the second conflicting block.

1) Theoretical Double Linking Process: In figure 2 the concept of a double linked BC is illustrated. Every block, expect the genesis block, can either be in the storage *Blocks With Tx* or *Blocks Without Tx*. This two versions of a block are indicated with *block-name*$_{w/}$ (including transactions) and *block-name*$_{w/o}$ (without transactions, meaning this block never contained transactions or all of them are aggregated by now). The increasing block number indicates which block is the ancestor, and the arrows point to them.

Every block, expect the genesis block, can contain transactions. These transactions are sent from clients to the network, what is indicated on the left side, as *incoming Tx*. Transactions are read as $FundsTx_{senderAddress=>receiverAddress}$ or when aggregated as $AggTx_{senderAddress=>receiverAddress}$. This happens as a historical aggregation in block 103 or while mining a new block denoted as the incoming *AggTx* also included in block 104.

As $FundsTx_{B=>D}$ reaches the network, the miners search in already validated blocks for other *FundsTx* or *AggTx* with either the same sender or receiver. In this example,

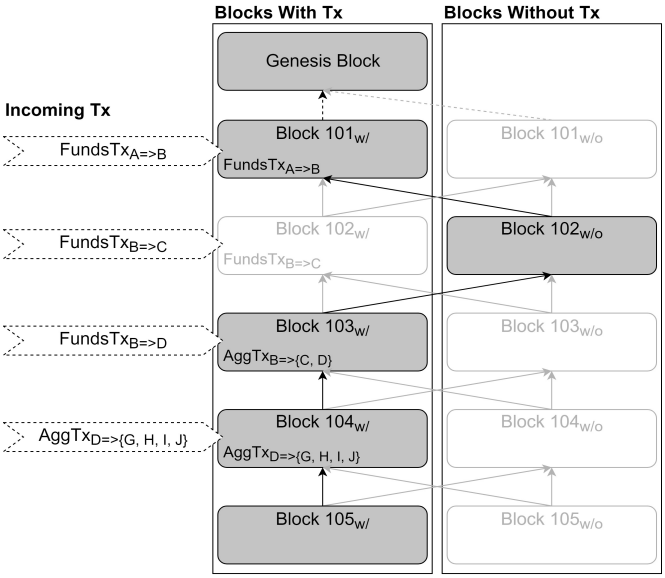


Fig. 2: Double Linked Blockchain Concept in Test-BC

$FundsTx_{B=>C}$ in block 102 can be aggregated, which leads to the case where in block 102 all transactions are aggregated.

Once all transactions are aggregated and the block is out of the exclusion zone, it can be transferred to the storage without transactions. The exclusion zone is defined as the *current blockheight* minus *NO_EMPTYING_LENGTH* what ensures that the user-defined *NO_EMPTYING_LENGTH* last blocks are not moved even though all their transactions are aggregated. Meaning only blocks with a blockheight smaller than *currentBlockheight - NO_EMPTYING_LENGTH* are moved. Block 105 is not emptied yet despite the fact it does not contain any transactions. Once block 105 will be out of the exclusion zone, it will be transferred to the *Blocks Without Tx*. When a *NO_EMPTYING_LENGTH* of *2* is assumed, block 105 can be moved once a block with height 108 is appended to the chain.

When emptying block 102, it will be moved to the *Blocks Without Tx*, and the hash *HashWithoutTransactions* gets valid. Therefore block 103 is not linked to block 102 via the *Previous Hash* anymore but over the *PrevHashWithoutTransactions* now. It has to be ensured that one link between two consecutive blocks is always valid. In figure 2 this is indicated with the black arrows between blocks. This results in a valid chain indicated with grayish background color whereas all other blocks are only there for visual purposes and therefore slightly faded.

III. Performance Evaluation

Evaluations of Test-BC is performed by setting up a test-bed including 20 miners hosted on Amazon Web Services (AWS) located in different countries around the globe. In this network more than 2,200 transactions monitored between the clients and miners. The maximal block size is set to 500,000 Bytes. The block interval, which defines the time span between two blocks added to the chain is set to 180 s and the difficulty interval is set to 20 blocks. This means that the difficulty is checked all 20 blocks and if blocks are mined too fast or too slow the target gets adapted accordingly to match the defined 180 seconds in future. Related information can be seen in Table I.

A. Performance Analysis Metrics

The metrics used in the evaluation process in this work are based on formulated metrics in [8] and defined as follows.

TPS: Transaction Per Second Represents the average number of transactions validated per second calculated with the help of the timespan between the first transaction sent to the network and the first block which includes all sent transactions, called the *timespanValidation*. The *TPS* on its own does only tell how many transactions are validated in a BC per second. It does not reveal anything about limiting factors and why a specific *TPS* is reached.

$TPS_{calc.}$ The maximal possible calculated number of transactions, which can be validated with a set block size and interval, is computed out of the number of transactions which fit into a block and the block interval (in seconds). The blockSize unit is byte and provides the upper maximum when blocks are validated in the correct user-set interval. Therefore, it is mainly used as a benchmark, since it is basically just a theoretical value. It should always be handled with care because the theoretical and not the actual block interval is taken into account for this. It indicates the maximal speed a BC can have without any type of scalability improvements, such as sharding or transaction aggregation. However, if the *TPS* is very close or similar to the $TPS_{calc.}$ and the TPS_{sent} is far above than the other two, the BC is probably limited either through the block size or by the block interval.

TPS_{sent} The *sent transactions per second* is the average number of transactions sent to the network by all clients and it indicates the upper limit for the number of transactions which can be validated per second. The TPS_{sent} indicates how fast transactions are sent to the network. When the TPS_{sent} and the *TPS* are close to each other, it can be assumed, that all transactions get validated shortly after they are issued. This because it does not take much longer to validate all transactions than it takes to send them. Thus, if they diverge a lot, it is an indicator, that it takes longer until the transactions are validated. Furthermore, it is an indicator for the upper *TPS* limit, since the *TPS* can not be higher than the TPS_{sent}.

ABI: Average Block Interval The *ABI* does reveal the actual timespan between two consecutive blocks. It does indicate if the network satisfies the defined block interval. Adjusting the block interval needs time to become consistent. It is simply measured as the difference between two following blocks or as an average with the timespan between two selected blocks (*endTime - startTime*) and the number of blocks between them.

BCS The *BCS* is the metric for the BC's overall size. The BCS consists out of the fixed part and the size of all transactions. Since Test-BC only writes the transaction hashes into blocks, the actual transaction size is not taken into account here. Once they get emptied, the overall BC size should shrink.

TABLE I: Analysis of Different Block Size and Transaction Aggregation on BC Scalability

Block size		Transactions							
Defined	ABS	#sent	#validated	TPS_{sent}	TPS	TPS_{min}	TPS_{max}	$\frac{TPS}{TPS_{calc.}}$	Transaction Aggregation
1'000	342	179'328	178'196	33.1	28.0	14.6	32.7	39.2	Enabled
5'000	4'342	181'933	181'933	33.3	28.5	20.9	33.2	3.1	Enabled
20'000	19'324	181'613	181'613	33.0	28.0	16.7	32.9	0.7	Enabled
1'000	342	19'000	19'000	36.3	0.6	0.6	0.6	0.7	Disabled
5'000	4'342	74'960	74'834	34.7	7.0	7.0	7.0	0.8	Disabled
20'000	19'342	181'078	181'078	33.5	27.3	27.2	27.4	0.7	Disabled

B. Performance Analysis Scenarios

Performance analysis of Test-BC based on the introduced metrics are performed on 20 miners running on AWS and GCPs located on various locations in the world. These analysis are performed while considering the effect of block size, interval between blocks, and the BC's overall size on the three test cases defined as following.

1) without transaction aggregation,
2) with transaction aggregation enabled,
3) with transaction aggregation & the emptying of blocks enabled.

1) Evaluation on Different Block Sizes: Here the evaluation and comparison between different block sizes and its influence on the *TPS*, with an initially defined amount of transactions sent to the network, is measured and listed. Table I shows the TPS rates for test cases with/without transaction aggregation.

As presented in the Table I, the block size does not take a big influence on the reached *TPS* value. This can be explained from multiple reasons:

Transaction Aggregation: Since transactions are aggregated, not every transaction needs space in a block, and thus, more transactions fit into one block. Whether there are two transactions from *A* to *B* or if there are one hundred transactions from *A* to *B*. With transaction aggregation, only one transacting will be written into the block.

Unlimited *AggTx* Size: If a transaction does not fit in one block, it is likely that it is validated in the next block because there is no limit in how many transactions can be written into one *AggTx* and because of the *splitAndSort*-algorithm's design. This algorithm takes the senders or receivers which have most awaiting transactions to be validated. Thus, the more transactions from one sender or to one receiver are in the mempool, the more likely it is that they get validated. Because the transactions from/to one wallet, which cannot be validated in a block, are still in the mempool, it is likely that there are more transactions matching the selection criteria for the next block. Since the size of an *AggTx* is unlimited, they are able to aggregate as many transactions as possible.

Test Case: It is likely, that the same sender or receiver is found quickly. This does help to keep the *TPS* at a high level. Transaction aggregation works better the more similar sender or receivers are in the transactions.

Transaction Sending: Transactions are sent approximately every 0.5 second. If transactions would only be sent after the previous transaction is validated in the network, aggregating by sender would not work. The current acknowledgment a miner sends to a client is only confirming that a certain transaction is sent to the network. A client does not know if and when the sent transaction is validated. Thus, aggregating by sender is possible.

The $\frac{TPS}{TPS_{\text{calc.}}}$ indicates by which factor the aggregation increases the maximum possible $TPS_{calc.}$ value. Thus with a block size of 1'000 byte, the version with transaction aggregation can theoretically handle roughly 39 times more transactions than without aggregation. It is clear, that with a smaller $TPS_{calc.}$ and a constant *TPS*, this factor increases. The $\frac{TPS}{TPS_{\text{calc.}}}$ for a block size of 20'000 byte is below one, because theoretically, over 600 transactions fit into a 20'000 byte block what results in a $TPS_{calc.}$ of about 40 transactions. This $TPS_{calc.}$ is higher than the TPS_{sent} and therefore, the ratio cannot be bigger than one.

As a preliminary conclusion, it can be stated that it is not possible to generalize, that with transaction aggregation it is exactly 39.2 respectively, 3.1 times faster because these numbers are highly influenced by the actual *TPS* which can not be higher than the TPS_{sent}. Thus, sending more transactions probably increases the *TPS*, and therefore also the $\frac{TPS}{TPS_{\text{calc.}}}$ ratio, even more. In Table I, the fraction $\frac{TPS}{TPS_{\text{calc.}}}$ can be understood as a degree of utilization in terms of the defined and actual block size.

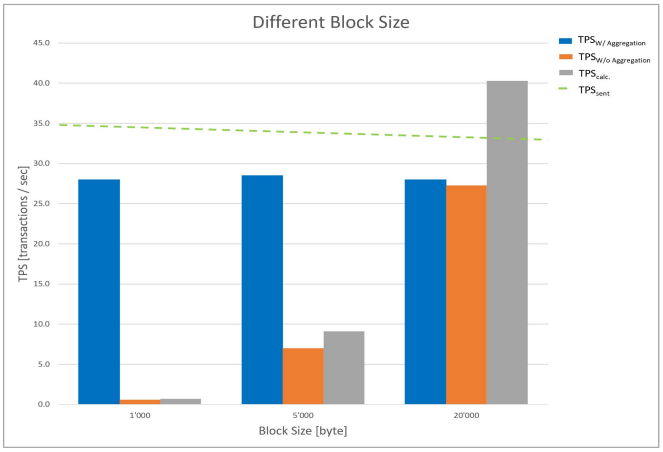


Fig. 3: Different Block Sizes influence on The *TPS*

Figure 3 visualizes the output of tables I and I. The scalability improvement is clearly visible. One can say, that with transaction aggregation, the block size is not a limiting factor anymore if there are either same senders or same receivers in the transactions. This is visible in figure 3, because TPS

TABLE II: Analysis of Different Block Intervals and Transaction Aggregation on BC Scalability

Block interval		Transactions							
Defined	ABI	#sent	#validated	TPS_{sent}	TPS	TPS_{min}	TPS_{max}	$\frac{TPS}{TPS_{calc.}}$	Transaction Aggregation
15	18.8	181'933	181'933	33.3	28.5	20.9	33.2	3.1	Enabled
60	66.8	190'000	190'000	34.8	34.4	33.9	34.7	15.3	Enabled
120	118.4	181'278	181'278	33.8	32.9	30.2	33.4	28.5	Enabled
15	14.3	74'960	74'834	34.7	7.0	7.0	7.0	0.8	Disabled
60	63.5	78'375	78'375	36.3	2.0	2.0	2.0	0.9	Disabled
120	111.4	18'000	18'000	34.4	1.1	1.1	1.1	1	Disabled

with aggregation (the blue bars) are nearly constant, whereas test runs without aggregation (orange bars), are increasing with bigger block sizes. The grey bars visualize the $TPS_{calc.}$. Furthermore, in test runs without aggregation do not reach to higher TPS in the test run with a block size of 20,000 byte. The block's size is not limiting the *TPS* anymore, since the $TPS_{calc.}$ is higher as the TPS_{sent}. This may indicate that the connection issues are limiting the network throughput.

The TPS_{sent} is indicated by the green line. The shrinking of the TPS_{sent} in figure 3 is not related to the block size, as it may could be assumed. Sending the transactions, even with a fixed interval, is still not always equally fast. It is related to the timespan in which a client receives an acknowledgment from a miner after sending the transaction. This timespan can slightly differ, and thus, it can have a bigger influence when sending a lot of transactions.

In the test runs with a block size of 1,000 Byte, the version with transaction aggregation can handle that much more transactions because about 10 transactions fit into one block. These 10 transactions can be aggregation transactions aggregating way more transactions and therefore increasing the *TPS*. Here point two from the listing above does have an influence because 19 different wallets are in the network. When aggregating these transactions perfectly, it would result in 19 *AggTx* transactions. This would overflow the block size by nine transactions. If transactions from one sender cannot be validated in block *n*, there will be all these transactions plus the newly received ones for block *n + 1*. Thus, during the preparation of block *n + 1*, this specific sender will have more transactions than a sender whose transactions get validated in block *n* and therefore all its transactions get validated then.

The block size is not limiting the Test-BC version with transaction aggregation up to the point, where only distinct senders and receivers are sending and receiving the transactions. Therefore, in regards to the block size, transaction aggregation does increase the *TPS*, especially for small blocks.

2) Different Block Intervals: Here the comparison and evaluation between different block intervals, with a given number of transactions sent to the network, is measured and listed.

Table II shows the *TPS* rates for test cases with/without transaction aggregation enabled. The block interval calculated in *seconds*, and the values belonging to transactions have units of *transactions* or *transactions per second*. The fraction $\frac{TPS}{TPS_{calc.}}$ can be seen as an improvement factor in contrast to the theoretical maximum and in table II as degree of utilization.

The *ABI* (Actual Block Interval) is an indicator if the BC validates blocks in the user defined interval. It is in *second* units and indicates the average timespan between two blocks. Since the validation speed is set with the help of the *target*, it is not exactly the set interval. This *target* is adapted every *n* blocks, whereas *n* is user defined.

The test runs with different block intervals look similar to the test runs with various block sizes. The *TPS* can be increased when transaction aggregation is used. Especially because the block interval and the *TPS* without aggregation are behaving inversely proportional, these test runs show the advantages. The relationship between the *TPS* and the block interval is inversely proportional because, with a higher block interval, fewer blocks can be validated, and thus, less transaction as well. The block size, on the other side, is related proportionally to the *TPS*, because larger blocks can handle more transactions. This results in a higher *TPS* value. Consequently, the version without aggregation can theoretically handle the most transactions with a tiny block interval and huge blocks.

Similar to the different block sizes, the block interval is not a *TPS* limiting factor. Transaction aggregation allows validating more transactions per second. During one test run, with a block interval of 60 seconds, the *TPS* is close to the TPS_{sent}.

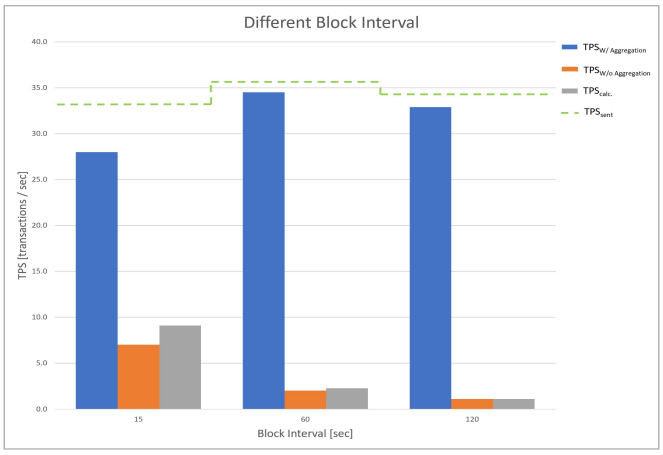


Fig. 4: Different block intervals and its influence on the *TPS* with and without aggregation

In figure 4 the differences of the TPS_{sent} were relatively big and thus it has different heights. The differences are again caused by small differences between sending a transaction and

receiving the acknowledgment from a miner. The difference between the *TPS* with aggregation (blue bars in Fig. 4) and without (orange bars) is even bigger here, since the block interval and the *TPS* are inversely proportional.

It is not certainly valid to conclude that with a higher block interval the *TPS* can be increased for Test-BC with aggregation enabled. Theoretically, the different block intervals should not have an effect on the *TPS* up to a certain number of different senders and or receivers. This is because with the two higher block intervals, four to eight times fewer blocks have to be sent through the network compared to the default 15 seconds interval. Therefore, the miners probably disconnect less often and they have more time to fetch transactions or blocks, which they never received.

3) Blockchain's Overall Size: As shown in Fig. 5, the BC size can be reduced with aggregation and even more with aggregation and emptying of blocks. The difference between only aggregating and aggregating with emptying is not big, because, in this test case only three different transactions are written into a block with aggregation. When emptying a block, the block's size gets reduced only around three times the size of a transaction hash. The more different transactions are listed in a block, the larger this difference will be.

When Test-BC with transaction aggregation is compared to a version without aggregation, the difference is sensible because with transactions aggregation enabled, around three transactions are validated in each block, whereas without aggregation, roughly 135 transactions get aggregated in each block. These 135 transactions are also the maximum capacity of a block with the defined size of 5,000 Byte. The two major escalations are caused by the start and end of sending transactions, whereas the smaller ones are caused by rollbacks. This graph shows the possibility of having a smaller overall BC size with transaction aggregation and emptying of blocks.

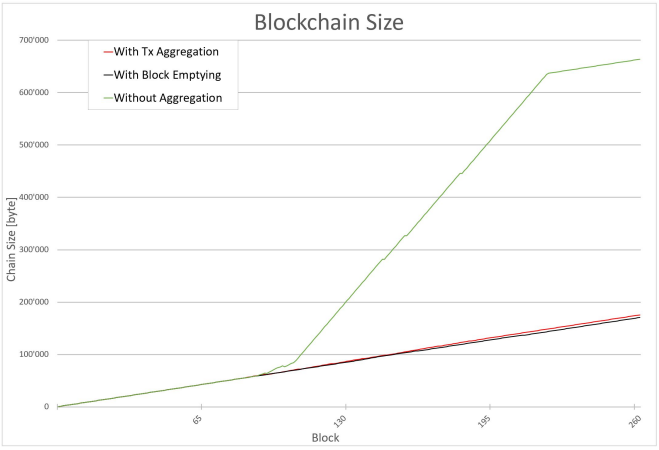
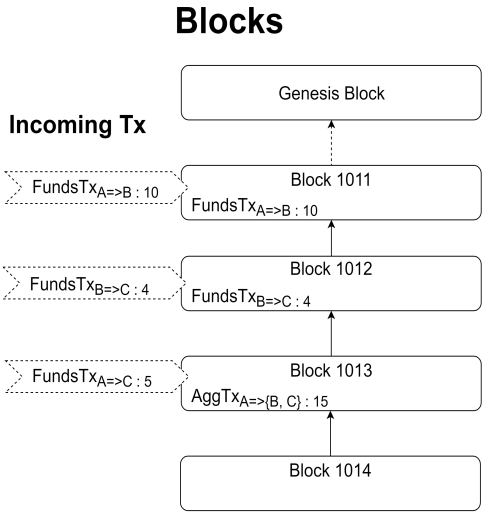


Fig. 5: Test-BC Size Comparison with 1)Aggregation, 2)Aggregation and Emptying of Blocks, and 3)Without Aggregation.

IV. Obstacles Of Transaction Aggregation

The intention behind double linking the BC is emptying all blocks once they are secure enough. A block is secure enough when it is accepted by the majority of miners, and thus, will not be included in rollbacks anymore. The emptying helps to save storage however, the emptying of validated blocks is kind of a contradiction against the core concept of a BCs, where secure blocks are immutable and cannot be changed again. Thus, some impediments occur, especially when a new miner joins the network and wants to start mining.

1) Joining As A New Miner & Order Of Transactions Issue: In figure 6 three FundsTx are incoming to the BC and get validated in blocks *1011*, *1012* and *1013*. As it is visible, the third transaction ($FundsTx_{A=>C:5}$) can be aggregated with the first transaction ($FundsTx_{A=>B:10}$) because of the similar sender. This results in the $AggTx_{A=>\{B,C\}:15}$ in block *1013*, and the removing of $FundsTx_{A=>B:10}$ in block *1011*.
The table on the right side shows the balances for the three wallets *A*, *B* and *C* with and without aggregation, before, between and after the three blocks are validated.



Wallet	Without	With
A	100	100
B	0	0
C	0	0
A	90	100
B	10	0
C	0	0
A	90	100
B	6	-4
C	4	4
A	85	85
B	6	6
C	9	9

Fig. 6: Transaction Aggregation and The Balance

As it is now visible in the tables in the right side of the Fig. 5, the balances for *A* and *B* are not the same when aggregating the transaction as when not aggregating them. This can lead to problems, especially when restarting or joining the network after transactions are already sent. Since Test-BC fetches all blocks from the last validated one to the genesis block first and afterward validates them in the correct order, moving and aggregating transactions is problematic.

For instance, when the transactions $FundsTxtA => B : 10$ and $FundsTx_{A=>C:5}$ get aggregated to $AggTx_{A=>\{B,C\}:15}$ and thus transaction $FundsTx_{A=>B:10}$ moves from block *1011* to *1013*, *B* does not have enough funds for transaction $FundsTx_{B=>C:4}$ at the point of validating block *1012*. This is visible in the middle two sub-tables where the balance of *B* is not the same with and without aggregation. When a new miner is joining the network, which uses transaction aggregation, and

the $FundsTx_{A=>B:10}$ is not in Block *1011* but in *1013*, this new joining miner is not able to validate block *1012*, because *B* does not have enough funds.

One Way of eluding this issue could be a credit-like behavior on the startup. This concept allows a wallet to have a negative balance during the startup process. At the end, similar to the fourth small table in Fig. 5 the balances with aggregation enabled would be the same as when validating each transaction without aggregation. As long as all transactions are validated, the order of validation does not play a substantial role. At the participation of a miner, this new miner only validates transactions which are already validated from other miners in the network. If the bootstrap miner tries to send invalid transactions to the new miner, the currently joining miner will find them, either by invalid block hashes, or when other miners are refusing its mined blocks later. Consequently, with this credit like behavior during the participation, it is possible to validate block *1012* even with aggregation and join the network.

Furthermore, if sharding is implemented on the Test-BC, it can be assumed that new miners are only able to join when an *epoch block* is inserted to the chain. This is needed because of the load balancing and the division into shards. Because these *epoch block*s are similar for every miner, and thus every miner agrees on all blocks before this *epoch block*, joining is not problematic, even if the transactions are not validated in the correct order. Additionally, in case the self-contained proof mechanisms are merged in the Test-BC PoS protocol, and therefore not all transactions are needed to prove that a wallet has enough funds, the transactions before an *epoch block* will not be used anymore.

A. Joining As A New Miner & Nonce Issue

Joining as a new miner, when blocks are already emptied, is problematic since the nonce of a block is calculated with the help of the wallets' balances. Here, it is possible that blocks are not validated because the nonce is incorrect at this point. To prevent this issue, the nonce should not be checked on startup. It is also possible to argue, that these blocks are validated in the network already and therefore they are secure enough to be accepted without re-validation of transactions.

This problem probably will also minimized when other scalability methods such as sharding is merged, because then it is ensured that at the time an *epoch block* is inserted, the network agreed on one block. Thus all miners approve blocks up to this *epoch block* as valid. When only emptying blocks up to this *epoch block*, only commonly accepted blocks are emptied and the nonce of these blocks does not have to be checked during startup.

V. Conclusions

Transaction aggregation method designed for Test-BC does definitely allow more transactions in one block. Thus the overall throughput increases and at the same time the block size and the overall chain size can be kept small. Furthermore, the block interval can be enlarged, which reduces network traffic. However, transaction aggregation performs better the more transactions with the same sender or receiver are sent. Thus, for use cases such as IoT-integrated applications where many IoT nodes send transactions to one (or some) predefined receiver(s), transaction aggregation approach helps the most in scaling the BC.

Although transaction aggregation already works well here with the Test-BC, its full power can be sensed even more when hybrid scalability techniques such as sharding and self-contained proofs are also enabled. The issues with new comer miners and the order of transactions and nuance value can be circumvented if the credit-like approaches are employed or the security of the already validated transactions are accepted by new miners.

finally, it is foreseen that private PoS and BFT- based blockchains can benefit the most from transaction aggregation mechanisms due to the federated management of the private BCs in relatively more trusted environments than purely public ones.

References

[1] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, E. Sirer, D. Song, and R. Wattenhofer, "On Scaling Decentralized Blockchains," Vol. 9604, 02 2016, pp. 106–125.

[2] "*Bitcoin, Proof of Work*," https://en.bitcoin.it/wiki/Proof_of_work, [Last visit June 6, 2018].

[3] A. Narayanan, J. Bonneau, E. Felten, A. Miller, and S. Goldfeder, *Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction*. Princeton University Press, 2016. [Online]. Available: \url{https://books.google.ch/books?id=LchFDAAAQBAJ}

[4] T. B. Burkhard Stiller, *Smart Contracts - Blockchains in the Wings*. Tiergartenstr. 17, 69121 Heidelberg, Germany: Springer, Jan 2017, pp. 169–184.

[5] "*Sharding, Raiden, Plasma: The Scaling Solutions that Will Unchain Ethereum*," https://urlzs.com/78MPp, [Last visit September 15, 2019].

[6] W. Wang, D. T. Hoang, P. Hu, Z. Xiong, D. Niyato, P. Wang, Y. Wen, and D. I. Kim, "A Survey on Consensus Mechanisms and Mining Strategy Management in Blockchain Networks," *IEEE Access*, Vol. 7, pp. 22 328–22 370, 2019.

[7] M. Belotti, N. Božić, G. Pujolle, and S. Secci, "A vademecum on blockchain technologies: When, which and how," *IEEE Communications Surveys Tutorials*, pp. 1–1, 2019.

[8] B. Wang, S. Chen, L. Yao, B. Liu, X. Xu, and L. Zhu, *A Simulation Approach For Studying Behavior And Quality Of Blockchain Networks*. Springer, Cham, 06 2018, pp. 18–31.